

# Qu'est-ce que REST ?

- REpresentational State Transfer : terme introduit par Roy Fielding
- Ce n'est pas un protocole, un standard ou un format
  - il n'existe pas de spécification du W3C pour le décrire
- Ce n'est pas le remplaçant des services SOAP
  - ne propose pas les caractéristiques avancées telles que la qualité de service, les transactions, les files d'attentes, etc
- Il s'agit d'un style d'architecture, d'un "mode de compréhension du Web"

# La notion de ressource

- Un des concepts important
- Le web est composé de ressources accessibles à partir d'une URL
  - web : un ensemble de ressources et non pas de "pages web "
  - le web est une gigantesque collection de ressources
- Chaque ressource est accessible par une URI (Uniform Resource Identifier)
- Le client et le serveur communiquent en s'échangeant une représentation de la ressource
  - le format de cette représentation peut être du XML, du JSON, une image, un fichier vidéo, ...
  - un affichage HTML est juste l'une d'entre elles
  - une même ressource peut donc avoir plusieurs représentations
  - certaines ressources sont des collections d'autres ressources

# L'accès aux ressources

- Les ressources sont accessibles via un ensemble uniforme de commandes fourni par HTTP qui permettent de spécifier l'opération à effectuer sur une ressource
  - essentiellement GET, POST, PUT et DELETE
  - CRUD : Create, Read, Update, Delete
- La communication entre le client et le serveur : "sans état"
  - chaque requête du client vers le serveur doit contenir toutes les informations nécessaires pour que cette demande soit comprise
  - elle ne peut tirer profit d'aucun contexte stocké sur le serveur
  - l'état de la session est donc entièrement détenu par le client

# Les verbes



- Ce sont des actions qui sont applicables aux ressources
- Quatre verbes universels
  - GET pour retrouver une information
  - POST pour ajouter une nouvelle information
  - PUT pour mettre à jour une information
  - DELETE pour supprimer une information

# Appel des méthodes avec HTTP

- On peut appeler GET et POST avec HTTP
- On ne peut pas appeler PUT ou DELETE avec un navigateur
  - le client et le serveur doivent se mettre d'accord sur une convention à utiliser pour simuler les méthodes PUT et DELETE.
- Principale limitation : non standardisation des méthodes PUT et DELETE dans les navigateurs

# Conventions

- Toutes les ressources doivent être correctement identifiées, et de manière unique
  - chaque ressource devra se voir assigner une URL
  - l'URL en question devra être de la forme  
`http://www.site.com/contenus/1789`  
plutôt que  
`http://www.site.com/contenus.php?id=1789.`
- Les ressources doivent être catégorisées selon leurs possibilités offertes à l'application cliente
- Chaque ressource fera un lien vers les ressources liées
- La manière dont fonctionne le service sera décrite au sein d'un document WADL, ou simplement HTML

# Exemple

- La liste des jouets est disponible à l'URL suivante :
  - <http://www.youpilesjouets.com/jouets/>

- Le client reçoit une réponse sous la forme suivante :

```
<?xml version="1.0"?>
```

```
<p:Jouets xmlns:p="http://www.youpilesjouets.com/"  
          xmlns:xlink="http://www.w3.org/1999/xlink">
```

```
<Jouet id="0001"
```

```
  xlink:href="http://www.youpilesjouets.com/jouets/0001"/>
```

```
<Jouet id="0002"
```

```
  xlink:href="http://www.youpilesjouets.com/jouets/0002"/>
```

```
<Jouet id="0003"
```

```
  xlink:href="http://www.youpilesjouets.com/jouets/0003"/>
```

```
[...]
```

```
</p:Jouets>
```

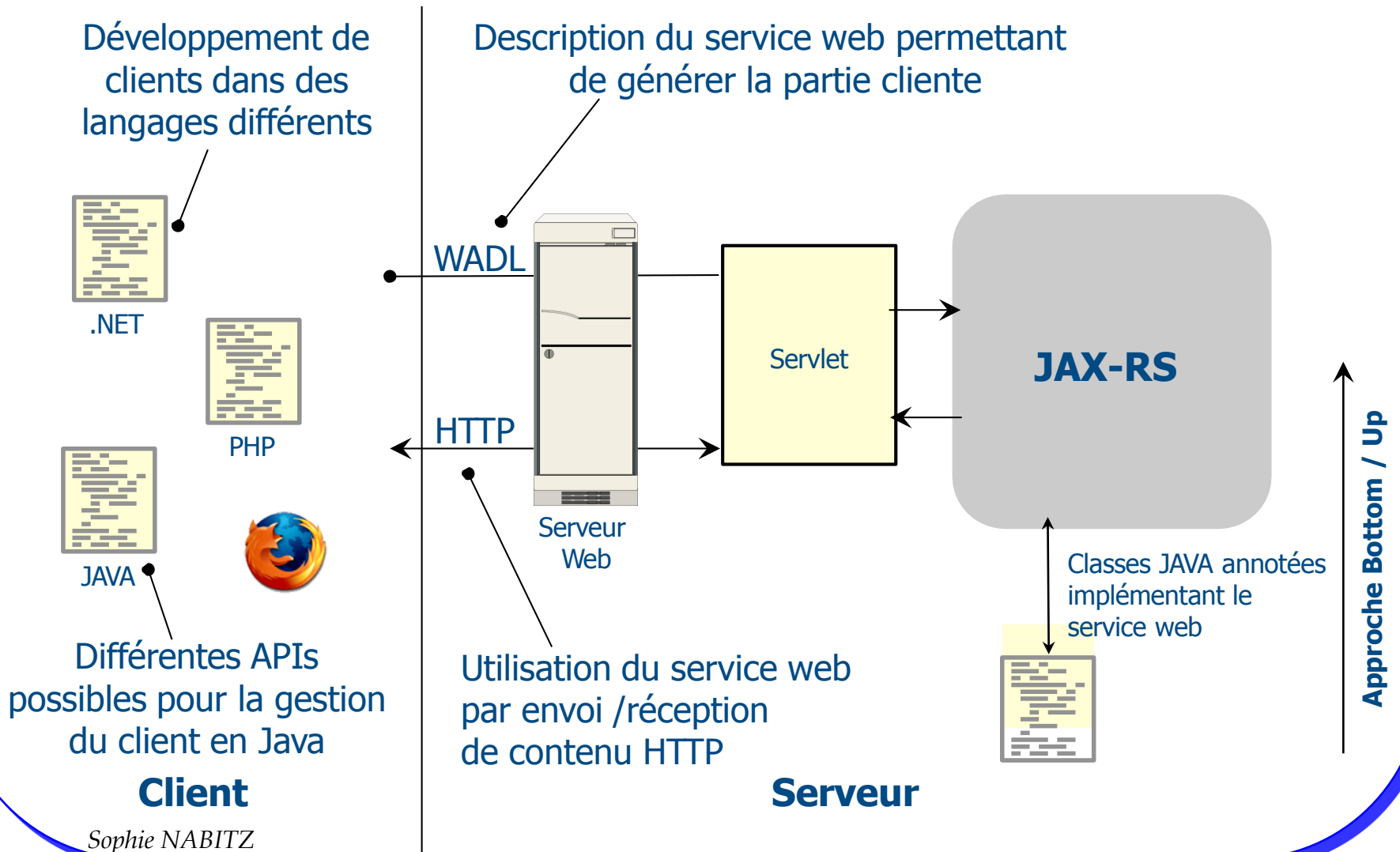
# JAX-RS

**La suite des transparents est totalement reprise des documents de M. Baron disponibles sur [developpez.com](http://developpez.com)**

- Java API for RESTful Web Services
  - mise en œuvre de services REST côté serveur et client
  - partie intégrante de la spécification Java EE 6
- Le développement des services REST repose sur l'utilisation de classes Java et d'annotations
- Implémentations JAX-RS
  - JERSEY : implémentation de référence fournie par Oracle
  - CXF : fournie par Apache, la fusion entre XFire et Celtix
  - RESTEasy : fournie par Jboss
  - RESTlet : un des premiers framework implémentant
  - WINK : implémentation fournie par Apache



# Fonctionnement



# Développement JAX-RS

- Basé sur des POJO (Plain Old Java Object) en utilisant des annotations spécifiques à JAX-RS
  - package javax.ws.rs
  - pas de description requise dans des fichiers de configuration
  - configuration de la Servlet JAX-RS requise pour réaliser le pont entre les requêtes HTTP et les classes Java annotées
- Un service REST est déployé dans une application web
  - un fichier de description est généré en WADL qui décrit les ressources disponibles dans l'application (le contexte)
  - Web Application Description Language basé sur XML

# Démarche

- Créer et annoter un POJO
- Compiler, déployer et tester
- Possibilité d'accéder au document WADL
- Le fichier de description WADL est généré automatiquement par JAX-RS
  - à consulter l'adresse `http://host:port/context/application.wadl`
- Approche Bottom / Up
  - pas de possibilité de développer un service REST à partir du fichier de description WADL

# Annotation @Path

- Une classe Java doit être annotée par @Path pour qu'elle puisse être traitée par des requêtes HTTP
- Cette annotation sur une classe définit des ressources appelées racines (Root Resource Class)
- La valeur donnée à @Path correspond à une expression URI relative au contexte de l'application web

`http://serveur:port/contexteAppliWeb/URIResource`

- Cette annotation peut également annoter des méthodes de la classe
  - l'URI résultante est la concaténation de l'expression du @Path de la classe avec l'expression du @Path de la méthode

# @Path : Template Parameters

- Possibilité de définir des expressions plus complexes appelées Template Parameters
  - la valeur dans @Path ne se limite pas seulement aux constantes
  - possibilité également de mixer dans la valeur de @path des expressions constantes et des expressions complexes
  - les Template Parameters peuvent également utiliser des expressions régulières
- Une expression complexe est délimitée par { ... }
  - exemple : @Path("{id}/editor")

# Paramètres de requêtes

- Annotations pour extraire les paramètres des méthodes
  - @PathParam : extraire les valeurs des Template Parameters
  - @QueryParam : valeurs des paramètres de requête
  - @FormParam : valeurs des paramètres de formulaire
  - @HeaderParam : paramètres de l'en-tête
  - @CookieParam : paramètres des cookies
  - @Context : informations liées aux ressources de contexte
- Valeur par défaut : @DefaultValue
- Exemple

```
@Path("{id}")
```

```
public String getBookById(@PathParam("id") int id) {  
    return "Java For Life "+ id;
```

```
}
```

# Méthodes HTTP



- L'annotation des méthodes Java permet de traiter de requêtes HTTP suivant le type de méthode
- Annotations disponibles

`@GET, @POST, @PUT, @DELETE`

- Uniquement sur des méthodes

# @Consumes et @Produces

- @Consumes : pour spécifier le(s) type(s) MIME qu'une méthode d'une ressource peut accepter
- @Produces : pour spécifier le(s) type(s) MIME qu'une méthode d'une ressource peut produire
  - exemple : `text/plain` : `String`
- Ces annotations portent sur une classe ou une méthode
  - l'annotation sur la méthode surcharge celle de la classe
- Si ces annotations ne sont pas utilisées tout type MIME peut être accepté ou produit
- Liste des différents types MIME dans la classe `MediaType`
  - exemple : `MediaType.APPLICATION_XML`



# Types personnalisés

- Le contenu d'une requête et d'une réponse peut être de l'XML ou du JSON
  - formes de contenu définies par @Produces et @Consumes
    - text/xml, application/xml, application/json
- Possibilité d'utiliser directement des types personnalisés en s'appuyant sur la spécification JAXB
  - spécification qui permet de mapper des classes Java en XML
  - avantage : manipuler directement des objets Java sans passer par une représentation abstraite XML
- Chaque classe est annotée pour décrire la correspondance entre le schéma XML et les informations de la classe
  - JAX-RS supporte la sérialisation/dé-sérialisation de classes annotées par @XmlRootElement, @XmlElement, @XmlType

# Injection de contexte

- L'annotation `@Context` permet d'injecter des objets liés au contexte de l'application
- `UriInfo` : informations liées aux URIs
- `Request` : informations liées au traitement de la requête
- `HttpHeaders` : informations liées à l'en-tête
- `SecurityContext` : informations liées à la sécurité

# @Context / HttpHeaders

- Un objet de type `HttpHeader` permet d'extraire les informations contenues dans l'en-tête d'une requête
- Principales méthodes
  - `Map<String, Cookie> getCookies()` : les cookies de la requête
  - `Locale getLanguage()` : la langue de la requête
  - `MultivaluedMap<String, String> getRequestHeaders()` : valeurs des paramètres de l'en-tête de la requête
  - `MediaType getMediaType()` : le type MIME de la requête
- Même résultat avec les annotations `@HeaderParam` et `@CookieParam`

# Réponses complexes

- JAX-RS permet aussi de retourner des réponses plus complexes qu'un type de base ou POJO/XML
- Les réponses complexes sont définies par la classe Response via des méthodes abstraites qui permettent
  - de choisir un code de retour
  - de fournir des paramètres dans l'en-tête
  - de retourner une URI, ...
- Méthodes sur la classe Response
  - Object getEntity() : corps de la réponse
  - T readEntity(Class<T> entityType)
    - *String* readEntity(*String.class*)
  - int getStatus() : code de retour

# Statuts des réponses

- Lors de l'envoi de la réponse un code statut est retourné
- Réponse sans erreur
  - les statuts des réponses sans erreur s'échelonnent de 200 à 399
  - code 200 "OK" pour les services retournant un contenu non vide
  - code 204 "No Content" pour les services retournant un contenu vide
- Réponse avec erreur
  - les statuts des réponses avec erreur s'échelonnent de 400 à 599
  - code de retour 404 "Not Found" : ressource non trouvée
  - code 406 "Not Acceptable": type MIME en retour non supporté
  - code 405 "Method Not Allowed" : méthode HTTP non supportée

# Response et ResponseBuilder

- Principales méthodes de la classe Response
  - pour créer une Reponse il faut utiliser des méthodes statiques qui retournent ResponseBuilder
    - utilisation du patron de conception Builder
  - ResponseBuilder created(URI location) : définit l'URI, à utiliser pour une nouvelle ressource créée
  - noContent() : pour une réponse vide (sans contenu)
  - ResponseBuilder ok() : statut à "Ok "
  - ResponseBuilder status(Response.Status) : pour un statut particulier
- Principales méthodes de la classe ReponseBuilder
  - Response build() : crée une instance
  - ResponseBuilder entity(Object value) : modifie le contenu du corps

# Déploiement

- Les applications JAX-RS sont construites et déployées sous le format d'une application web Java (WAR)
- La configuration de JAX-RS déclarer les classes ressources dans le fichier de déploiement (web.xml)
- Deux types de configuration sont autorisées
  - web.xml pointe sur une sous classe d'Application
    - la classe Application permet de décrire les classes ressources
  - web.xml identifie une servlet fournie par l'implémentation JAX-RS

# Descripteur de déploiement

```
<?xml version="1.0"encoding="UTF-8"?>
```

```
<web-app version="2.5"...>
```

```
<servlet>
```

```
<servlet-name>ServletName</servlet-name>
```

```
<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
```

```
<init-param>
```

```
<param-name>jersey.config.server.provider.packages</param-name>
```

```
<param-value>sn</param-value>
```

```
</init-param>
```

```
<load-on-startup>1</load-on-startup></servlet>
```

```
<servlet-mapping>
```

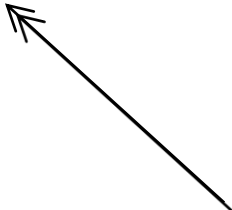
```
<servlet-name>ServletName</servlet-name>
```

```
<url-pattern>/*</url-pattern>
```

```
</servlet-mapping>
```

```
</web-app>
```

Servlet fournie par Jersey pour le traitement des requêtes HTTP





# Développement Client

- La spécification JAX-RS fournit l'API pour le traitement côté client
  - l'utilisation de l'API cliente ne suppose pas que les services web soient développés avec JAX-RS (.NET, PHP, ...)
  - package javax.ws.rs.client
- Avantages d'utiliser l'API cliente de JERSEY
  - manipuler les types Java (pas de transformation explicite en XML)
  - faciliter l'écriture des tests unitaires
- Démarche
  - initialiser et configurer le client
  - cibler une ressource

# Configurer le client

- Objet de la classe `javax.ws.rs.client.Client`
- Créer une configuration : objet de la classe `ClientConfig`
- Utiliser la méthode statique `newClient` de la classe `ClientBuilder`, en l'initialisant avec la configuration
- Créer la cible (application à atteindre) : `WebTarget`
- Préciser la cible au client : méthode `target`

# Création de la chaine cible

- La classe `WebTarget` fournit des méthodes pour construire l'en-tête de la requête
- Principales méthodes
  - `WebTarget path(String)` : définition d'un chemin
  - `WebTarget queryParams(String key, String val)` : paramètres
  - `Invocation.Builder request()`
- Possibilité d'appeler plusieurs fois la même méthode
  - exemple : `path`

# Création de la requête

- Démarche
  - création d'une chaîne d'appel de méthode dont le type de retour est WebTarget (path, queryParams...)
  - invocation d'une méthode dont le type de retour est un objet de la classe Invocation.Builder
  - utilisation des méthodes de Invocation.Builder correspondant aux verbes HTTP (GET, POST, ...)
- Méthodes pour accéder à la ressource
  - Response get()
  - `<T> get(GenericType<T> c)` : méthode GET avec un type de retour T
  - `<T> post(Entity<?> entity, GenericType<T> c)` : appelle la méthode POST en envoyant un contenu dans la requête
  - `<T> put(Entity<?> entity, GenericType<T> c)` : appelle la méthode PUT en envoyant un contenu dans la requête
  - `<T> delete(Entity<?> entity, GenericType<T> c)` : appelle la méthode DELETE en envoyant un contenu dans la requête JAX-RS